Prolog
et
Traitement Automatique des Langues

Éric de la Clergerie
Eric.De_La_Clergerie@inria.fr

ALPAGE – INRIA
http://alpage.inria.fr

Cours M2 LI 2008

Septième partie VII

Prolog comme analyseur

Quelques problèmes avec Prolog

Divers problèmes bien connus, dus à la gestion en profondeur d'abord avec retour-arrière de Prolog :

- terminaison sur des productions récursives
- non-complétude des réponses
- duplication importante de calculs

Terminaison

```
s --> gn, v, gn.
gn --> np.
gn --> gn, gp.
```

```
s L=[Paul,aime,Marie]
gn,v,gn L=[Paul,aime,Marie]
   gn L=[Paul,aime,Marie]
      np L=[Paul,aime,Marie]
         [Paul] L=[Paul,aime,Marie]
         v,gn L=[aime,Marie]
           v L=[aime,Marie]
             [aime] L=[aime,Marie]
               gn L=[Marie]
                  np L=[Marie]
                     [Paul] L=[Marie]     %fail
                     [Marie] L=[Marie]
                     [] L=[]
Answer:
    L = [Paul,aime,Marie]
```

Terminaison (suite)

```
gn,gp L=[Marie]
   gn L=[Marie]
      np L=[Marie]
         [Paul] L=[Marie]      % fail
         [Marie] L=[Marie]
            gp L=[]            % fail
      gn,gp L=[Marie]
         gn L=[Marie]
            np L=[Marie]
               [Paul] L=[Marie]  % fail
               [Marie] L=[Marie]
               gp L=[]           % fail
             gn,gp L=[Marie]
             ...
```

⇒ boucle de prédiction sur gn −−> gn,gp.

Complétude

```
s −−> gn, v, gn.
gn −−> gn, gp.
gn −−> det, nc.
```

```
s L=[Paul,aime,Marie]
  gn,v,gn L=[Paul,aime,Marie]
     gn L=[Paul,aime,Marie]
        gn,gp L=[Paul,aime,Marie]
           gn L=[Paul,aime,Marie]
              gn,gp L=[Paul,aime,Marie]
                 gn L=[Paul,aime,Marie]
                    gn,gp L=[Paul,aime,Marie]
                       ...
```

⇒ boucle, sans aucune réponse

Importance de l'ordre des productions :
Ennuyeux pour un formalisme déclaratif !
(cause : ordre non équitable de sélection des productions)

Boucles

Boucles et récursions en présence de récursions// en particulier sur les récursions gauches

$\text{NP} \leftarrow \text{NP Prep NP}$ (a) directe

$A \leftarrow B\ c \quad B \leftarrow A$ (b) indirecte

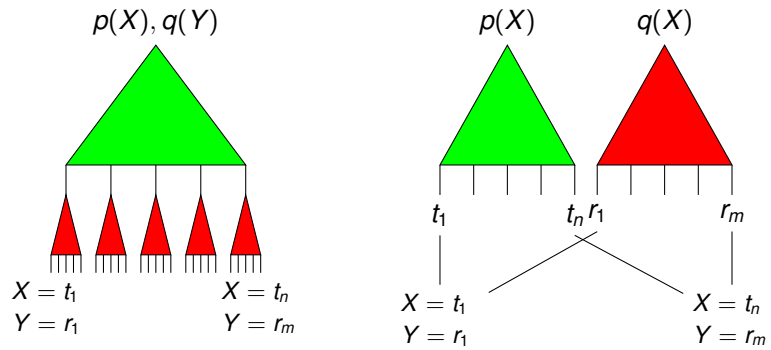
$A \leftarrow B\ A\ c \quad B \leftarrow \epsilon | b$ (c) cachée

Remèdes :

1. Transformation de programmes,
   - pas toujours possible ou facile
   - éventuellement incompatible avec des constructions sémantiques
   - pertube les arbres d'analyse attendus
2. Compare l'appel courant avec les appels en cours
   ⇒ mémo-fonction et tabulation
3. changer de stratégie d'analyse et de contrôle

Duplication de calculs

La gestion de non-déterminisme à la Prolog (retour-arrière) ⇒ beaucoup de re-calculs :



Sans partage, les GN sont recalculés 5 fois

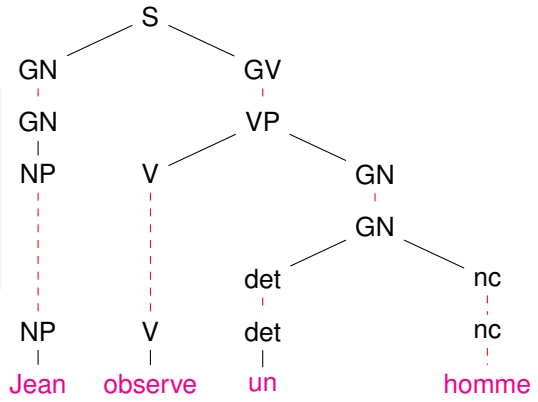
Jean observe un homme sur sur la colline avec un télescope.

Le nombre de re-calculs croit exponentiellement avec le nombre de points d'ambiguïtés.

Construire une analyse

Par collage d'arbres (partiels) d'analyse :

```
s  —> gn gv
gn —> np
gn —> det nc
gn —> gn gp
gv —> v np
gv —> gv gp
gp —> prep gn
```



Stratégies d'analyse

Une stratégie d'analyse décrit quels pas de calculs sont autorisés pendant l'analyse :

- stratégies descendantes (*top-down*), guidées par les buts, en partant de l'axiome
- stratégies ascendantes (*bottom-up*)
  guidées par les réponses, en partant des terminaux
- stratégies hybrides (comme la stratégie Earley)
- stratégies dirigées par des tables (Left Corner, Head Corner, LR, . . .)

Stratégies de contrôle

Une stratégie de contrôle spécifie la gestion du non-déterminisme, en particulier en terme d'ordonnancement :

Ambiguïtés Que faire des ambiguïtés

- désambiguïser (probabilités, heuristiques, regard en avant)
  désambiguïsation totale ou partielle
- exploration par retour arrière,
- exploration par tabulation, . . .

Ordonnancement Dans quel ordre examiner les alternatives ?

- en profondeur d'abord (dernière alternative examinée en premier)
- en largeur d'abord
- en fonction de probabilités ou heuristiques
- synchronisation lecture (gauche-droite) de la chaîne
- parallèle, concurrent, . . .

Huitième partie VIII

Implanter des stratégies d'analyse

Analyseur descendant

DCG Prolog $\equiv$ { analyse : descendante gauche-droite
contrôle : profondeur d'abord avec retour-arrière

```
phrase(A,L,R) :- td_parse(A,L,R).
td_parse(A,L,R) :-
        recorded((A --> G)),
        td_parse(G,L,R).
td_parse([],L,L).
td_parse([T|Rest],L,R) :-
        'C'(L,T,M),
         td_parse(Rest,M,R).
```

Facile à mettre en oeuvre ($\equiv$ descente récursive gauche [LL])
mais boucles, incomplétude et duplication de calculs.

En profondeur incrémentale

Même stratégie descendante, mais contrôle = profondeur d'abord mais bornée
ie :

- en profondeur d'abord, jusqu'à une certaine profondeur k
- en relançant avec une profondeur $k+1$

```
tdi_parse_iterate(A,L,R,Depth) :-
( tdi_parse(A,L,R,Depth)
; NewDepth is Depth+1,
  tdi_parse_iterate(A,L,R,NewDepth)
).
tdi_parse(A,L,R,Depth) :-
    Depth > 0, NewDepth is Depth-1,
    recorded((A --> G)),
    tdi_parse(G,L,R,NewDepth).
```

Permet d'éviter les incomplétudes dans les réponses, mais

- toujours des problèmes de terminaison
- encore plus de duplication de calculs

Analyseur ascendant : moteur

L'analyse part des terminaux pour remonter vers l'axiome
mise en oeuvre : réduction progressive de corps de production dans la chaîne

```
phrase(A,L,R) :-
     bu_parse(L,[A|R]).
bu_parse(L,L).
bu_parse(L,R) :-
       %% L = Front.Body.Tail
        split(L,Front,Body,Tail),
       %% trouve une prod Head --> Body
        dcg_match(Body,Head),
        split(M,Front,[Head],Tail),
       %% M = Front.[Head].Tail
        bu_parse(M,R).
```

```
:-std_prolog split/4.
split(L,A,B,C) :-
    append(A,BC,L),
    append(B,C,BC).
```


Analyseur ascendant : trace

```
bu [Paul,aime,Marie]
bu [np,aime,Marie]
bu [gn,aime,Marie]
bu [gn,v,Marie]
bu [gn,v,np]
bu [gn,v,gn]
bu [s]
Answer:
    L = [Paul,aime,Marie]
...
```

L'analyse termine, car aucune boucle de prédiction, mais ...

Analyseur ascendant : efficacité

Extrêmement inefficace !

- les constituants sont reconstruits dans tous les ordres possibles
- des constituants intermédiaires inutiles sont créés

```
Answer:
    L = [Paul,aime,Marie]
bu [gn,aime,np]
bu [gn,v,np]
bu [gn,v,gn]
bu [s]
Answer:
    L = [Paul,aime,Marie]
bu [gn,aime,gn]
bu [gn,v,gn]
bu [s]
Answer:
    L = [Paul,aime,Marie]
...
```


Analyseur shift-reduce : principe

Approche ascendante légèrement plus efficace :

- les terminaux sont lus de gauche à droite et empilés (shift) sur une pile
- réduction (reduce) quand le corps d'une production est présent sur la pile la tête de la production est empilée à la place

Peut être rendue plus efficace en pré-calculant des tables des `shift` et `reduce` autorisés dans un certain état :
⇒ stratégies LR et (non déterministe tabulaire) GLR (Tomita)

Analyseur shift-reduce : moteur

```
%% sr_parse(StackIn,StackOut,StringIn,StringOut)
sr_parse(SI,SI,L,L).

sr_parse(SI,SO,[A|M],R) :-
    sr_parse([A|SI],SO,M,R).

sr_parse(SI,SO,L,R) :-
    %% match en reverse le haut de la pile SI=[RevBody|SRed]
    %% avec le corps d'une clause Head --> Body
    sr_dcg_match(SI,Head,SRed),
    sr_parse([Head|SRed],SO,L,R).
```


Shift-reduce : Trace

```
shift Paul => [Paul]
shift aime => [aime,Paul]
shift Marie => [Marie,aime,Paul]
reduce to np => [np,aime,Paul]
reduce to gn => [gn,aime,Paul]%f
reduce to v => [v,Paul]
shift Marie => [Marie,v,Paul]
reduce to np => [np,v,Paul]
reduce to gn => [gn,v,Paul]%f
reduce to np => [np]
shift aime => [aime,np]
shift Marie => [Marie,aime,np]
reduce to np => [np,aime,np]
reduce to gn => [gn,aime,np]%f
reduce to v => [v,np]
shift Marie => [Marie,v,np]
reduce to np => [np,v,np]
reduce to gn => [gn,v,np]%
reduce to gn => [gn]
shift aime => [aime,gn]
shift Marie => [Marie,aime,gn]
reduce to np => [np,aime,gn]
reduce to gn => [gn,aime,gn]%f
reduce to v => [v,gn]
shift Marie => [Marie,v,gn]
reduce to np => [np,v,gn]
reduce to gn => [gn,v,gn]
reduce to s => [s]
Answer:
    L = [Paul,aime,Marie]
```

Plus efficace et terminaison, mais

- gestion explicite d'une pile
- toujours des re-calculs

Relation coin-gauche

Analyseur coin-gauche

Neuvième partie IX

Analyseurs tabulaires

Tabuler ?

Principe : garder des traces de calculs dans une table, pour

- détecter des boucles, afin d'améliorer la terminaison
- partager des sous-calculs
- déployer des stratégies d'ordonnancement plus flexibles
- extraire des traces des sous-calculs réussis (preuves, arbres d'analyse)
- agréger une valeur sur un ensemble de résultats (moyenne, somme, plus court chemin, . . .)

Tabulation et CFG

Une longue histoire avec de nombreux algorithmes :

- CKY [Cocke-Kasami-Younger]
- Algorithme d'Earley – Analyseurs à Charte [Kay]
- Generalized LR [Tomita]
- Automates à piles / programmation dynamique [Lang]

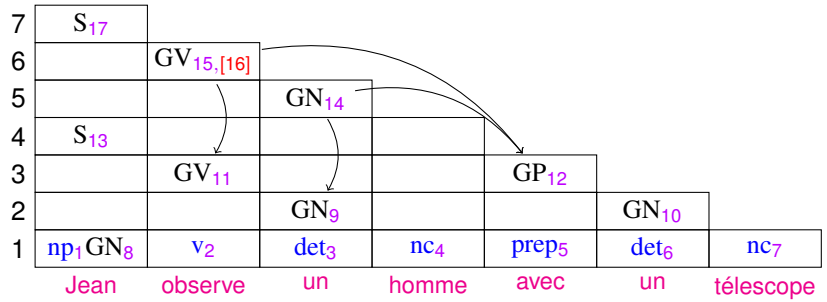
Mais essentiel de clairement distinguer

- Stratégie d'analyse
- Stratégie de contrôle (niveau tabulation)

Cocke-Kasami-Younger [CKY]

Algorithme en Programmation Dynamique (1965)
Analyse ascendante avec tabulation des constituants

> *Si il existe une production $A_0 \leftarrow A_1 \dots A_n$ avec, pour tout $i > 0$, A_i présent dans (x_i, l_i) et $x_{i+1} = x_i + l_i$, alors tabuler le non terminal A_0 dans l'entrée $(x_1, \Sigma_i l_i)$ (sauf si déjà tabulé).*



Les constituants généralement construits par longueur croissante,
de la gauche vers la droite
mais en fait non obligatoire !

Tabulation passive

- La table ne sert qu'à enregistrer des calculs terminés.
- La table ne sert pas à diriger les calculs

Complexité

```
table_initialize
for all positions x and lengths l
  for all productions A0 ← A1 ... Av
    for all lengths l1, ..., lv−1 with Σk=1..v−1 lk < l
          lv = l − Σk=1..v−1 lk
          xj = x + l1 + ··· + lj−1
          if Aj ∈ T[xj, lj] for all j > 1
          then add A0 in T[x, l]   (unless present)
```

Complexité temps pire des cas fournie par les itérations enchâssées sur x, l et l_j $(1 \leq j < v)$ bornées par la longueur de la chaîne n.
$\Rightarrow O(n^{v+1})$ où v est la longueur de la plus longue production

Pour un reconnaisseur, complexité espace pire des cas fournie par le # de cellules dans la table and le # de constituants par cellule
$\Rightarrow O(n^2)$

Forme normale de Chomsky (binarisation)

Complexité en $O(n^{v+1})$ réduite à $O(n^3)$ en mettant sous forme normale de Chomsky (**binarization**).

Règle ternaire VP --> V, NP, NP donne une complexité $O(n^4)$ mais peut être remplacée par les règles binaires

```
VP --> V, VP_ARGS.
VP_ARGS --> NP, NP.
```

Mais implique une transformation de la grammaire
plus élégant de manipuler des règles pointées

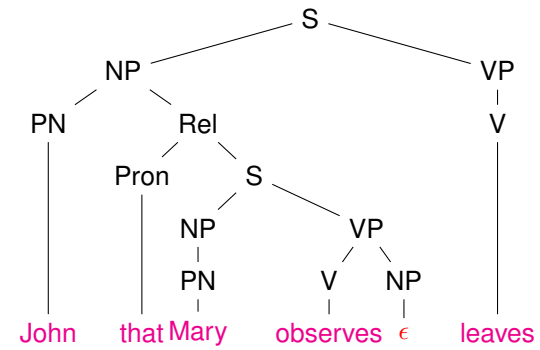
Pire des cas temps en $O(n^3)$ et espace en $O(n^2)$ (quasi) optimal pour les CFG mais CKY non efficace (défauts des stratégies purement ascendantes)

Limitations de CKY

Constituants inutiles

John who looks [$_S$ Marie leaves]

Traces



Analyses avec Charte

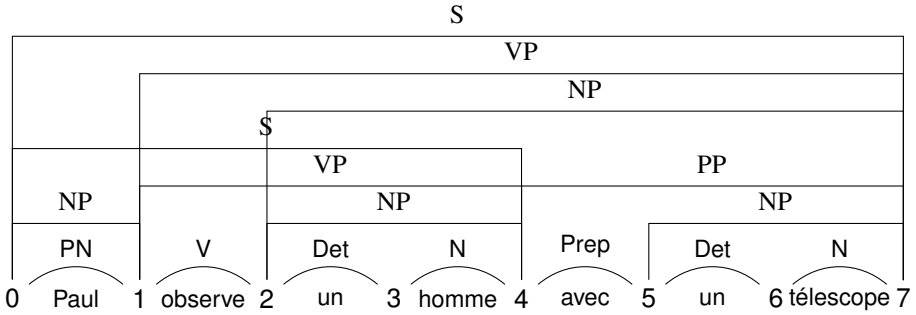
Historiquement, approches motivées par le souhait :

- d'utiliser de la tabulation (pour le partage de calculs)
- de préserver la complexité optimale $O(n^2)$ pour les CFGs
- d'introduire de la prédiction (descendante)

$\rightsquigarrow$ développement de techniques génériques fondées sur des chartes
une charte *equiv* ensemble d'arcs (items) entre 2 positions de la chaîne, avec divers types de labels

CKY reformulé en charte passive

Les entrées de la table CKY visuellement représentés par des arcs et stockés comme items $\langle i, j, Cat \rangle$.



Complexité temps en $O(n^{v+1})$

Interpréteur Prolog

Règles pointées

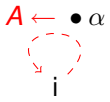
Une charte active stocke non seulement des constituants incomplets mais aussi des constituants partiels, pouvant servir à guider l'analyse.

Utilisation de

- règles pointées [*dotted rules*] $A_0 \leftarrow A_1 \ldots A_i \bullet A_{i+1} \ldots A_n$
- arcs étiquetés par des règles pointées (items $\equiv \langle i, j, A \leftarrow \alpha \bullet \beta \rangle$)
- un système déductif spécifie comment dériver les items

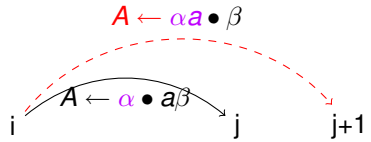
Un système déductif pour CKY

$$\frac{}{\langle i, i, A \leftarrow \bullet\, \alpha \rangle} \quad \exists A \leftarrow \alpha$$



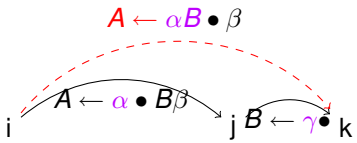
(Seed)

$$\frac{\langle i, j, A \leftarrow \alpha \bullet a\beta \rangle}{\langle i, j+1, A \leftarrow \alpha a \bullet \beta \rangle} \quad a = a_{j+1}$$



(Scan)

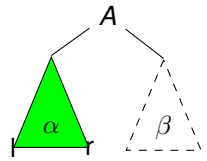
$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle \quad \langle j, k, B \leftarrow \gamma \bullet \rangle}{\langle i, k, A \leftarrow \alpha B \bullet \beta \rangle}$$



(Complete)

Invariant

Chaque item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ vérifie l'invariant : $\alpha \rightarrow^\star a_{l+1} \ldots a_r$

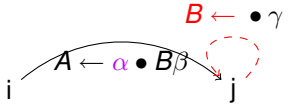


L'utilisation des règles pointées induit une binarisation implicite
$\Rightarrow$ complexité temps $O(n^3)$

Algorithme d'Earley

Possibilité d'ajouter une règle de prédiction (descendante)
⇒ Algorithme d'Earley [1970]

$$\frac{\langle i, j, A \leftarrow \alpha \bullet B\beta \rangle}{\langle j, j, B \leftarrow \bullet \gamma \rangle} \quad \exists B \leftarrow \gamma$$



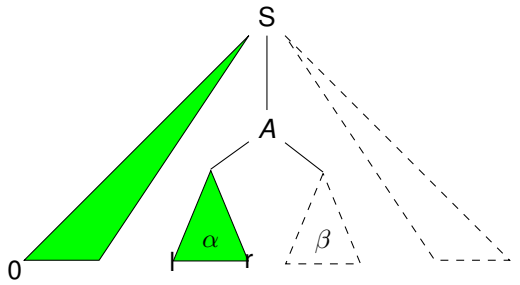
(Pred)

+ rules (Scan) and (Complete)

Invariant et complexité

Chaque item $\langle l, r, A \leftarrow \alpha \bullet \beta \rangle$ satisfait 2 invariants :

1. Reconnaissance de α entre l et r (comme pour CKY)
2. validité des préfixes : $\exists \gamma \in (T \cup N)^*,\ S \rightarrow^* a_1 \ldots a_l A \gamma$



La complexité pire des cas en temps demeure $O(n^3)$
Mais, en pratique, la prédiction réduit l'espace de recherche et réduit la complexité

Charte : mise en oeuvre

Un algorithme à charte repose sur :

- une table (i.e. charte) où sont stockés les items, sans duplication.
- un agenda où sont stockés les items à traiter

Un cycle de l'algorithme implique

1. Sélectionner un item I dans l'agenda
2. Si I n'est pas déjà tabulé, l'ajouter ; sinon retour étape 1
3. Construire de nouveaux items en combinant I avec les items déjà tabulés
4. Insérer les nouveaux items dans l'agenda

Variante : Les items sont d'abord tabulés avant insertion dans l'agenda
Ordre de sélection (Earley) : $\langle i, j, A \rangle$ choisi avant $\langle k, l, B \rangle$ si $j < l$:
⇒ synchronisation gauche-droite de la lecture

Pour les CFG, l'ordre de sélection importe peu (univers fini) :
⇒ l'algorithme termine et est complet

Formulation Prolog

Un item (A −−> Body)(I,J), en notation Hilog
Variante de Earley : règle pointée $A \leftarrow \alpha \bullet \beta$ représentée par $A \leftarrow \beta$

```
earley_parse(A,L,R) :-
  predict(A(L,L),[],Agenda),
  process(Agenda),
  stored((A --> true)(L,R)).
```

```
process([]).
process([Item|OldAgenda]) :-
  process_item(Item,OldAgenda,Agenda),
  process(Agenda).
```

- process traite le 1er item de l'agenda
- stored test si l'item est dans la table

Formulation Prolog (process)

```
process_item((B-->true)(J,K),OldAgenda,Agenda) :-
    resolve_passive(B(J,K),OldAgenda,Agenda).

process_item(Item::(A-->B,Beta)(I,J),OldAgenda,Agenda) :-
    predict(B(J,J),OldAgenda,MiddleAgenda),
    resolve_active(Item,MiddleAgenda,Agenda).
```


Formulation Prolog (predict)

Prédit de nouvelles production à réduire.

```
predict(G::A(I,I),Agenda1,Agenda2) :-
  all_solutions(NewItem,
                G^prediction(G,NewItem),
                Agenda1,
                Agenda2).
prediction(A(I,I),NewItem) :-
   recorded(Prod::(A-->Body)),
   store(NewItem::Prod(I,I)).
```

- all_solutions : collecte tout les items résultant de la prédiction et les ajoute à l'agenda
  (Prolog : findall , bagof ou setof, DyALog : iterate , . . .)
- store ajoute l'item à la table ; échoue si déjà présent

Formulation Prolog (passive)

Utilise un fait pour réduire des règles pointées

```
resolve_passive(Fact::B(J,K),Agenda1,Agenda2) :-
  all_solutions(NewItem,
                Fact^passive(Fact,NewItem),
                Agenda1,
                Agenda2).

passive(B(J,K),NewItem) :-
   stored((A -->B,Beta)(I,J)
   store(NewItem::(A->Beta)(I,K)).
```


Formulation Prolog (active)

À partir d'une règle pointée, cherche un fait tabulé pour avancer

```
resolve_active(Item::(A-->B,Beta)(I,J),Agenda1,Agenda2) :-
  all_solutions(NewItem,
                Item^active(Item,NewItem),
                Agenda1,
                Agenda2).

active((A-->B,Beta)(I,J),NewItem) :-
   stored((B -> true)(J,K))
   store(Item::(A->Beta)(I,K)).
```

findall (Prolog)

```
habite(jean, belfort).
habite(lucie, paris).
habite(christian, toulouse).
habite(adeline, paris).
habite(nicolas, paris).

?- findall(X, habite(X,paris), R).
R = [lucie,adeline,nicolas] ?
```


Schéma d'analyse

Description of parsing strategies in terms (of classes) of partial parse trees

> *[Sikkel]* *"These intermediate results are not necessarily partial trees, but they must be objects that denote relevant properties of those partial parses."*

A schema indicates

- the domain of items (and their form)
- the item invariants

Very close from chart algorithms

Stratégie par coin-gauche

Stratégie dirigée par les têtes

Construire un arbre d'analyse

Earley revisité

Test de redondance

Boucles : variance vs sumsomption

Subsomption faible et forte

Terminaison et spirales

Grammaires finiment ambiguës

Boucles de prédiction

Couper les boucles de prédiction

Restrictions

Implantation des restrictions en Prolog

Restrictions en DyALog

Ordonnancement et complétude

Dixième partie X

Efficient tabular parsing strategies

Efficient strategies

Tabulation and parsing strategies are two largely orthogonal issues.

But some strategies are more efficient than others when using tabulation

These strategies may differ from those used for deterministic evaluations.

Useless argument

Building and propagating parse trees reduce computation sharing

$$\mathrm{VP} \leftarrow \mathrm{VP}\ \mathrm{PP} \rightsquigarrow \mathrm{VP}(vp(VP, PP)) \leftarrow \mathrm{VP}(VP)\ \mathrm{PP}(PP)$$

$$\langle i, j, \mathrm{VP}(t)\rangle + \begin{cases} \langle j, k, \mathrm{PP}(t_1)\rangle & \rightarrow & \langle i, k, \mathrm{VP}(vp(t, t_1))\rangle \\ \langle j, k, \mathrm{PP}(t_2)\rangle & \rightarrow & \langle i, k, \mathrm{VP}(vp(t, t_2))\rangle \end{cases}$$

$\Rightarrow$ duplication of computations and no search space pruning

More judicious to extract a tree (or forest) from tabulated traces

Similar problem for semantic forms

Example of modulation : hanoi

Idée : oublier pendant les prédiction les arguments non-prédictifs

```
% hanoi with accumulator list
hanoi([],_,_,_,M,M).
hanoi([X|Y],L,C,R,M1,M2) :-
        hanoi(Y,C,L,R,M3,M2),
        hanoi(Y,L,R,C,M1,[m(L,R)|
    M3]).
```

Pour partager sur les plots & accumulateur :

	Call	Return
hanoi(stack, l, c, r, moves, acc)	call_hanoi_6(stack)	ret(*l*, *c*, *r*, moves, acc)

$$\left.\begin{array}{l}\text{hanoi}(Y, C, L, R, M3, M2)\\ \text{hanoi}(Y, L, R, C, M1, [m(L, R)|M3])\end{array}\right\} \rightsquigarrow \text{call\_hanoi\_6}(Y) \quad \Rightarrow \quad \text{Sharing}$$

En DyALog : :−mode(hanoi/6,+(+,−,−,−,−,−).